A PARALLELIZED ITERATIVE CLOSEST POINT ALGORITHM FOR ATTITUDE ESTIMATION

A Thesis

by

AMBER SUE JARRATT

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	John Hurtado
Co-Chair of Committee,	Srinivas Vadali
Committee Member,	Thomas Ioerger
Head of Department,	Rodney Bowersox

August 2016

Major Subject: Aerospace Engineering

ABSTRACT

In recent decades, low-earth orbital debris has become a major concern. If the density of objects in orbit is too high, and nothing is done to remove any debris, there could be a cascade of collisions, each generating more debris, increasing the frequency of collisions even further. This would render future space missions very difficult and dangerous, if not impossible. Something must be done to remove space debris from orbit. This thesis attempts to solve one piece of the orbital debris problem - that is, tracking a piece of debris and determining its attitude and position relative to a capture vehicle. A LIDAR camera is used to acquire images of the body to be tracked. To achieve a fast and practical solution, a parallelized Iterative Closest Point (ICP) algorithm and a Kalman filter are implemented to track the attitude and position of a model rocket booster. Additionally, this thesis presents a method to increase ICP's accuracy and reliability by artificially increasing image resolution by algorithmically increasing image size. This work also explores the performance of different variations of ICP and the dependency of their performance on image size.

I dedicate this thesis to my husband, Travis Jarratt,

without whom this work would not be possible.

ACKNOWLEDGEMENTS

I would like to thank all of my committee members for their support as well as Dr. Suman Chakravorty and Dr. Manoranjan Majji for their guidance. I would also like to thank NASA JSC, Vector-Nav Technologies, and Systems & Processes Engineering Corporation for their funding to make this research possible.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER I

INTRODUCTION

I.A. Motivation

Orbital debris is an unfortunate reality of space travel and poses a major threat if nothing is done to alleviate the problem. More than 20 years ago, the number of objects in orbit of a size greater than 1 centimeter was estimated to be about 118,000. Of these, about 8,000 are greater than 10 centimeters in size [1]. Figure I.1 shows an orbital debris plot by NASA [2]. The white dots in the image represent space objects that are being tracked, 95 % of which are non-functional satellites.

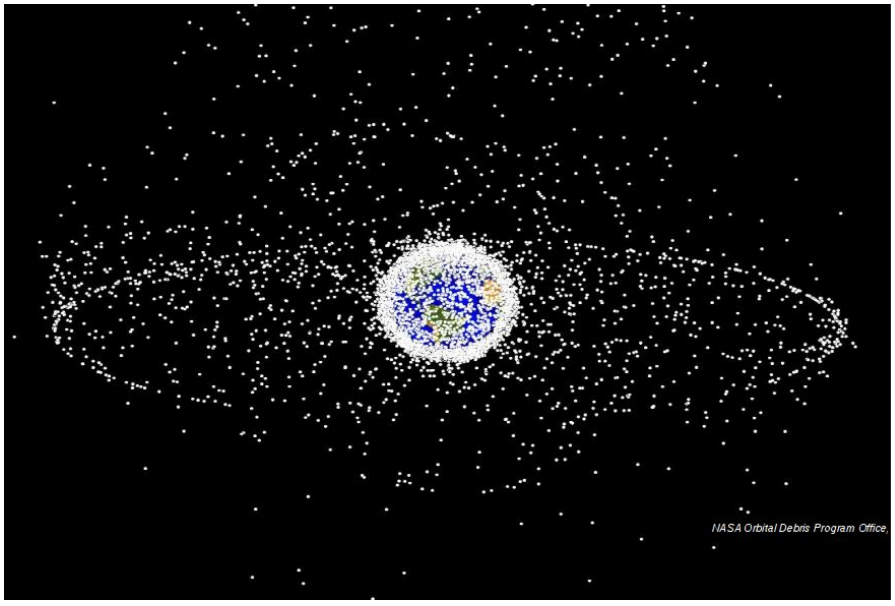

Figure I.1. Orbital debris [2]

I.B. Overview of Iterative Closest Point

The name "ICP" was mentioned for the first time in 1992 by Besl and McKay [3]. They described the problem as follows:

"Given 3-D data in a sensor coordinate system, which describes a data shape that may correspond to a model shape, and given a model shape in a model coordinate system in a different geometric shape representation, estimate the optimal rotation and translation that aligns, or registers, the model shape and the data shape minimizing the distance between the shapes"

This can be achieved by implementing a mean-squared-error metric. There are multiple ways to do this, which will be explained in Chapter III.

The research done towards this thesis has been focused on autonomous space debris removal applications. The iterative closest point algorithm is well suited for this problem, given a CAD model of the object of interest. With this algorithm, it is possible to estimate the pose (position and attitude) of the object, which is essential for any proximity operation and useful in many other applications. The model used in this thesis is that of a Russian SL-8 rocket booster. The CAD model of this rocket booster is shown in Figure I.2.

A multiple step process is required to transform the CAD file into a format usable by ICP. This format is essentially a list of vertices and their corresponding normal vectors. First, the CAD file is exported as an STL file, which converts the file to a set of vertices and triangles. Then, the STL file is converted to a PLY file. This was achieved though an open source software called "MeshLab" [4]. With some minor changes, this PLY file is essentially what the ICP algorithm in this thesis accepts. This file is shown as a point cloud in Figure I.3 in 4 different views.

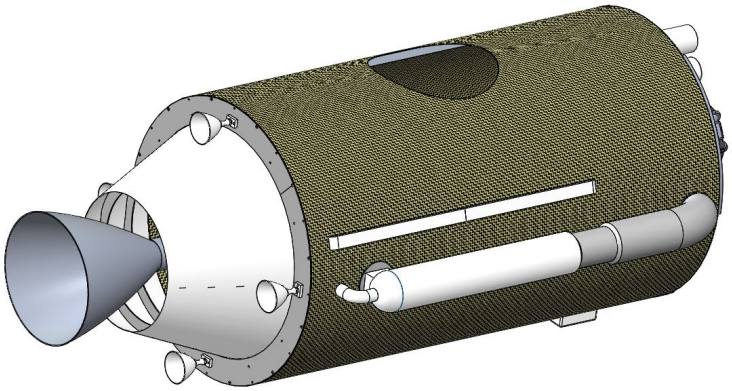

Figure I.2. CAD model of the SL-8

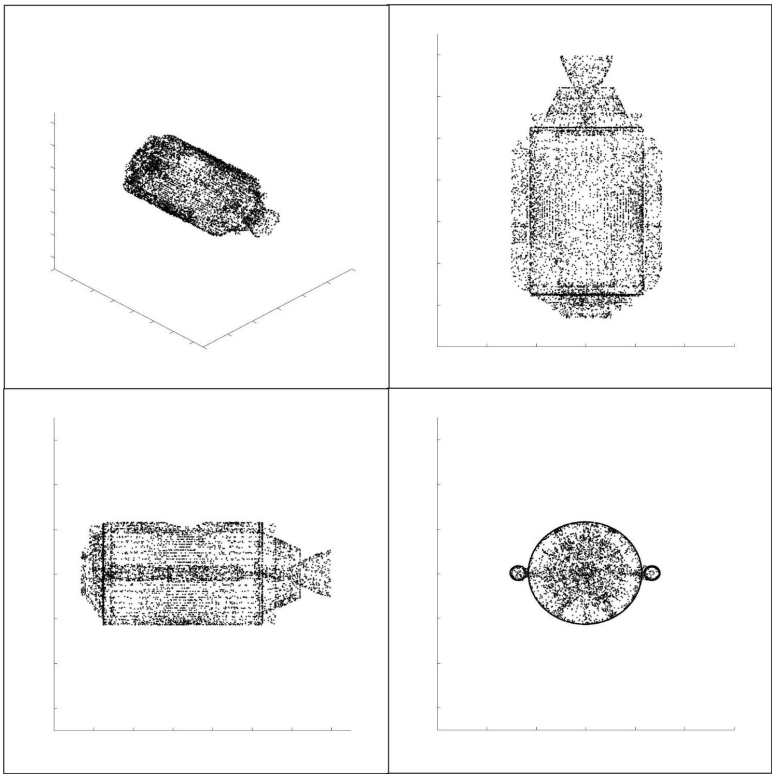

Figure I.3. Point cloud of the SL-8

The ICP algorithm presented in this thesis solves for the pose (attitude and position) that transforms the body from sensor coordinates to model coordinates. A visualization module was developed to help verify the convergence of the algorithm. Figure I.4 shows a screenshot of the algorithm in action. The image on the left is the image of the rocket booster directly from the sensor. The image on the right is the model point cloud (the same as Figure I.3) transformed into sensor coordinates using the pose estimate computed from the ICP algorithm. These two images should match fairly closely — if they don't, it is an indication that ICP has not converged.

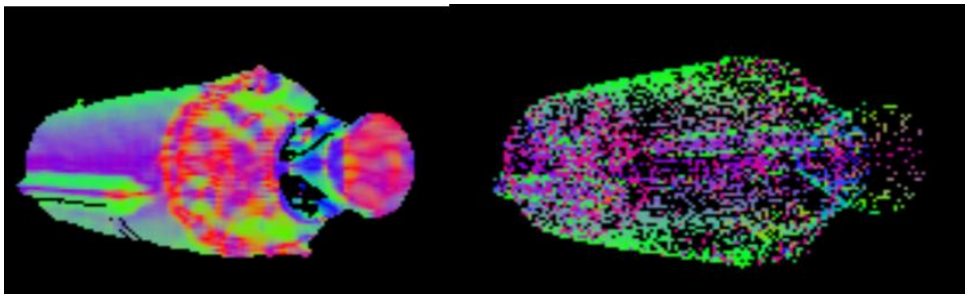

Figure I.4. ICP visualization module

The ICP algorithm has its limitations. This algorithm is only applicable if one has a CAD model (or something similar) of the object of interest. In addition, the algorithm is somewhat computationally intensive. This thesis presents a way of parallelizing this algorithm in III.C to combat this issue.

I.C. Reference Frames

Figure I.5 shows how the reference frames are defined for this research problem. The desired attitude is the rotation matrix from the sensor frame to the target frame. The position that is desired is the translation from the sensor to the target in the target frame. In this thesis, "target" and "model" are synonymous. The IMU (inertial measurement unit) frame becomes relevant when obtaining attitude data from the IMU to be used in a Kalman filter.

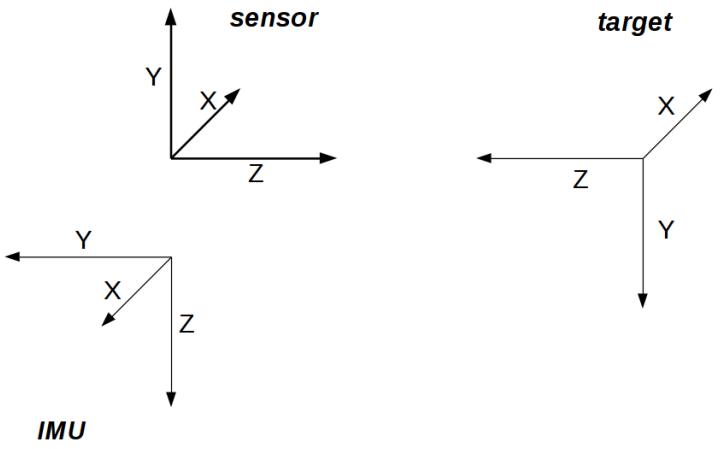

Figure I.5. Relevant reference frames.

I.D. Thesis Structure

This thesis begins by explaining in Chapter II the processing involved before any LIDAR images are able to be used by the ICP algorithm and presents a method of algorithmically increasing image resolution. Additionally, this chapter also covers the correspondence problem and illustrates its importance to this research study.

Chapter III details the ICP algorithm and derives its two variants. This chapter also offers a way to parallelize ICP and outlines the manner in which the algorithm is integrated into the overall debris capture system. Results obtained by implementing the concepts in the previous two chapters are presented in Chapter IV. Chapter V examines the effect of the addition of a Kalman Filter to the system. Chapter VI concludes this thesis by reviewing potential areas that may require additional research.

CHAPTER II

THE CORRESPONDENCE PROBLEM AND IMAGE PROCESSING

In order to understand the Iterative Closest Point algorithm well, it is first necessary to understand how and why the images must be processed before being used by the algorithm.

II.A. Coordinate-Pixel Mapping

II.A.1. Intrinsic Camera Parameters

For the ICP algorithm to work, a pixel-to-coordinate and coordinate-to-pixel correspondence must be known. The former is easy to deal with — it is simply a matter of accessing data directly from the sensor. However, the latter is not as simple. One must construct a camera model and obtain the intrinsic parameters. This allows for the computation of pixel coordinates given any {x,y,z} coordinate in the image. To accomplish this, the Camera Calibration Toolbox for Matlab by the California Institute of Technology [5] was used to obtain the sensor parameters.

In the camera model, there are 10 different calibration parameters: focal length (2x1), principal point (2x1), skew coefficient (1x1), and distortion coefficients (5x1). Only 9 of these were used in the calibration because the skew coefficient was found to be negligible. The definition and usage of these parameters is described below.

$$\text{focal length} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \tag{2.1a}$$

$$\text{principal point} = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{2.1b}$$

$$\text{distortion coefficients} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix} \tag{2.1c}$$

The goal is to find the pixel coordinates that correspond to a given spatial coordinate. First, we find the normalized (pinhole) projection, p_n, using Equation 2.2. The quantities X, Y, and Z are the physical spatial coordinates in the camera frame.

$$p_n = \begin{bmatrix} X/Z \\ Y/Z \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.2}$$

Now, we include lens distortion to the model. The distorted normalized coordinate, p_d, is given by Equation 2.3. The quantities r and dp are defined in Equation 2.4 and Equation 2.5, respectively.

$$p_d = \begin{bmatrix} p_{d,1} \\ p_{d,2} \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_5 r^6) p_n + dp \tag{2.3}$$

$$r^2 = x^2 + y^2 \tag{2.4}$$

$$dp = \begin{bmatrix} 2k_3xy + k_4(r^2 + 2x^2) \\ k_3(r^2 + 2y^2) + 2k_4xy \end{bmatrix} \tag{2.5}$$

Finally, the pixel coordinates, u and v, can be computed using Equation 2.6.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x p_{d,1} + u_0 \\ f_y p_{d,2} + v_0 \end{bmatrix} \tag{2.6}$$

II.A.2. Building a Look-up Table

To avoid doing the calculations outlined in the previous subsection each time a pixel location is required, a look-up table is constructed at the start of the program before any image is even taken. This is possible because the camera parameters and the coordinate-to-pixel mapping is independent of any one image. Building a look-up table takes advantage of the array's constant-time access, thereby reducing computation time on each frame. Algorithm 1 shows the process of building the correspondence look-up table. In the algorithm, the variable X_n will range from values of X_0 to X_f, incrementing by δx. The width of the table, W, is equal to the total number of values X_n steps though. Similarly, Y_n will range from Y_0 to Y_f, incrementing by δy. The height of the table, H, is equal to the total number of values Y_n steps though. The constants in GetPixel are the same camera parameters described in the previous section.

Algorithm 2 shows how to use the correspondence look-up table once it has been created. The variable *table* is assumed to be a global array, and the parameters X_0, Y_0, δy, and δx in Algorithm 2 are the same as those in Algorithm 1. This XyzToPixel function is applicable for any image taken from the camera.

Algorithm 1 Building the Correspondence Table

procedure BUILDTABLE($X_0, \delta x, Y_0, \delta y$)
  for i = 0 to H-1 **do** ▷ table height
    for j = 0 to W-1 **do** ▷ table width
      $X_n \leftarrow X_0 + j\ \delta x$
      $Y_n \leftarrow Y_0 + i\ \delta y$
      $\{U, V\} \leftarrow$ GETPIXEL(X_n, Y_n)
      $table(i, j, 0) \leftarrow U$
      $table(i, j, 1) \leftarrow V$
    end for
  end for
end procedure

function GETPIXEL(X_n,Y_n)
  $R \leftarrow X_n^2 + Y_n^2$
  $u = X_n(1 + k_1R + k_2R^2 + k_5R^3) + 2k_3X_nY_n + k_4(R + 2X_n^2)$
  $v = Y_n(1 + k_1R + k_2R^2 + k_5R^3) + 2k_4X_nY_n + k_3(R + 2Y_n^2)$
  $U \leftarrow$ ROUND($f_u u + u_0$)
  $V \leftarrow$ ROUND($f_v v + v_0$)
  return $\{U, V\}$
end function

Algorithm 2 Accessing the Table

function XYZTOPIXEL(X, Y, Z)
  $X_n \leftarrow X/Z$
  $Y_n \leftarrow Y/Z$
  $j \leftarrow$ ROUND($(X_n - X_0)/\delta x$)
  $i \leftarrow$ ROUND($(Y_n - Y_0)/\delta y$)
  $U \leftarrow table(i, j, 0)$
  $V \leftarrow table(i, j, 1)$
  return $\{U, V\}$
end function

II.B. Image Processing

The image obtained by the sensor cannot be used in its raw form due to the high level of noise. The camera used in this work provides a range measurement (from which a three-dimensional coordinate can be inferred) for each pixel in the image. The device will report a measurement for a pixel even if the measurement is so uncertain that it is useless. Because of this, some processing must be done to get the image in a workable state. A raw image from the sensor is shown in Figure II.1. This is an image of the SL-8 model rocket booster approximately 2 meters away from the camera. From this image, it is nearly impossible to distinguish the object from the sensor noise.

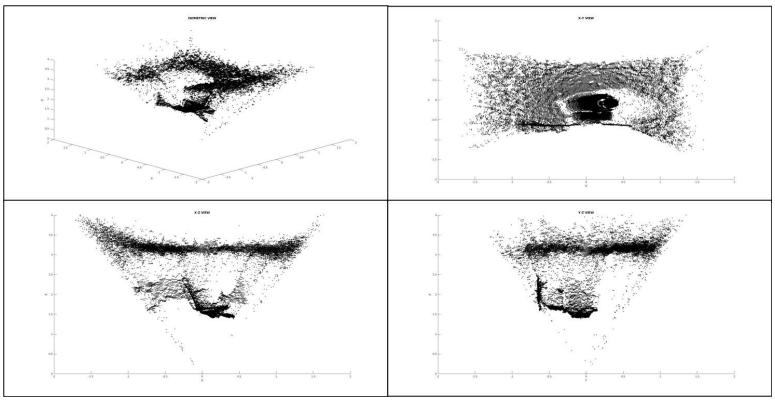

Figure II.1. Raw sensor data. Top left: isometric view. Top right: X-Y view. Bottom left: X-Z view. Bottom right: Y-Z view.

II.B.1. Image Thresholding

Fortunately, not only does the sensor supply a range reading for each pixel, it also provides a confidence level for each individual measurement in the form of a 16-bit unsigned integer. This information is used to nullify measurements that don't meet a certain confidence level. In addition to the confidence threshold, a simple physical coordinate bounding box is used to filter the data. The filtered version of Figure II.1 is shown in Figure II.2. The pseudocode for image thresholding is shown in Algorithm 3

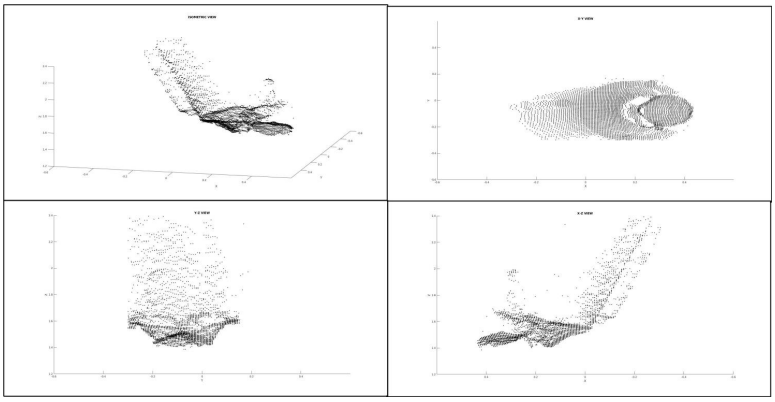

Figure II.2. Filtered sensor data. Top left: isometric view. Top right: X-Y view. Bottom left: X-Z view. Bottom right: Y-Z view.

II.B.2. Bilateral Filter

Bilateral filters are typically used to smooth gray and color images, as in [6]. In a bilateral filter, each pixel intensity is transformed as a weighted sum of neighboring pixel intensities. Importantly, these weights depend on the difference in the pixels' intensities as well as the difference in the pixels' coordinates. The "intensities" can

Algorithm 3 Image Thresholding

procedure PROCESSIMAGE($Image, \{X_l, X_u\}, \{Y_l, Y_u\}, \{Z_l, Z_u\}, c$)
  for each pixel $\{U, V\} \in Image$ **do**
    $\{X, Y, Z\} \leftarrow Image.xyz[\{U, V\}]$ ▷ xyz coordinate at pixel $\{U, V\}$
    $C \leftarrow Image.conf[\{U, V\}]$ ▷ confidence value at pixel $\{U, V\}$
    if $X \notin [X_l, X_u]$ **or** $Y \notin [Y_l, Y_u]$ **or** $Z \notin [Z_l, Z_u]$ **or** $(C < c)$ **then**
      $Image.xyz[\{U, V\}] \leftarrow 0$
    end if
  end for
end procedure

be anything from RGB values, depth values, or Cartesian coordinates. In the case of this thesis, the intensities referring to Cartesian coordinates. A mathematical description of a bilateral filter is shown in Equation 2.7. In this equation, $X(p)'$ is the updated Cartesian coordinate at pixel p and $X(p)$ is the current Cartesian coordinate at pixel p.

$$X(p)' = \frac{1}{W} \sum_{p_i \in \Phi} X(p_i) w(p, p_i) \tag{2.7}$$

In Equation 2.7, $X(p)'$ is the updated Cartesian coordinate at pixel p that we wish to compute and $X(p)$ is the previous value at that same pixel. The term Φ represents the set of neighboring pixels to be used in the filter, and $X(p_i)$ is the Cartesian coordinate at one of the pixels in that neighborhood. The weights $w(p, p_i)$ and W are defined in Equation 2.8 and Equation 2.9, respectively.

$$w(p, p_i) = f_x(||X(p_i) - X(p)||) g_p(||p_i - p||) \tag{2.8}$$

$$W = \sum_{p_i \in \Phi} w(p, p_i) \tag{2.9}$$

The functions given by f_x and g_p can in general be any distribution desired. In this work, these functions are both Gaussian. The function f_x weighs points based

on the difference in Cartesian coordinates between the point we are updating and that of the particular pixel in the summation. On the other hand, g_p weighs points based on the difference in pixel coordinates. Let the pixel p be represented by the pair (i, j), and let a particular pixel p_i in Φ be defined by the pair (k, l). A visual summary of these quantities is shown in Figure II.3.

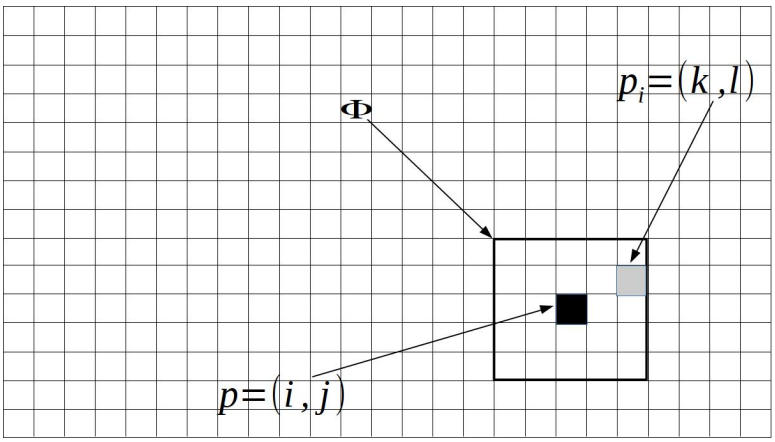

Figure II.3. Visual representation of bilateral filter quantities

With these definitions in mind, the weight functions can be defined as in Equation 2.10

$$f_x = e^{-((i-k)^2+(j-l)^2)/2\sigma_x^2} \tag{2.10a}$$

$$g_p = e^{-||X(i,j)-X(k,l)||^2/2\sigma_p^2} \tag{2.10b}$$

The terms σ_x and σ_p are smoothing parameters. Increasing σ_x puts more emphasis on the importance of physical coordinate proximity, whereas increasing σ_p puts a higher emphasis on the importance of pixel coordinate proximity. Algorithm 4 gives the pseudocode for the bilateral filter.

Algorithm 4 Bilateral Filter

function BILATERALFILTER($Image, \sigma_x, \sigma_p, h$)
  $X \leftarrow Image.xyz$
  for each pixel $p \in Image$ **do**
    $W \leftarrow 0$
    $x \leftarrow 0$
    $\{U, V\} \leftarrow p$
    $\Phi \leftarrow$the set of pixels from $\{U-h, V-h\}$ to $\{U+h, V+h\}$
    for each pixel $p_i \in \Phi$ **do**
      $w_i \leftarrow (e^{-||p-p_i||^2/2\sigma_p^2})(e^{-||X(p)-X(p_i)||^2/2\sigma_x^2})$
      $W \leftarrow W + w_i$
      $x \leftarrow x + w_i\ X[p_i]$
    end for
    $X[p] \leftarrow x/W$
  end for
  $Image.xyz \leftarrow X$
  return $Image$
end function

This bilateral filter is used in two ways in this thesis. First, in addition to spatial coordinates, the ICP algorithm requires a normal map of each image frame. To produce a smoother normal map, a bilateral filter is applied to the image before normals are computed at each pixel. The filtered image is only used to produce this map, but it is not used otherwise. Second, the bilateral filter is used in the process of increasing the resolution of the images acquired by the camera. This is explained in the next subsection.

II.B.3. Increasing Image Resolution

A naive (and rather useless) way to increase image resolution is to expand each pixel by a given scaling factor. This would produce a "stair effect" and won't actually improve the image resolution. In order to be useful, a bilateral filter with relatively high smoothing parameters is applied to the expanded image. Crucially, not every pixel is expanded. Expanding a noisy image could have disastrous consequences, or at best case be completely useless. Instead, certain pixels are strategically selected to be

expanded — that is, pixels with high confidence values as described in Section II.B.1. This allows the edges of the object in the images to be sharper and better defined. The pseudocode for this method of enlarging images is presented in Algorithm 5.

Algorithm 5 Enlarge Image

```
function EnlargeImage(Image, s, c, σ_x, σ_p, w)
    X ← Image.xyz
    C ← Image.conf
    Image'.xyz ← 0
    Image'.conf ← 0
    for each pixel p ∈ Image do
        {U, V} ← p
        u ← s U
        v ← s V
        if C[p] > c then
            for i = 0 to s − 1 do
                for j = 0 to s − 1 do
                    Image'.xyz[{u + i, v + j}] = X[p]
                    Image'.conf[{u + i, v + j}] = C[p]
                end for
            end for
        else
            Image'.xyz[{u, v}] = X[p]
            Image'.conf[{u, v}] = C[p]
        end if
    end for
    Image' ← BilateralFilter(Image', σ_x, σ_p, w)
    return Image'
end function
```

Figure II.4 shows how the confidence acceptance level for pixel expansion affects the resulting image. At a confidence level of 50%, all the points in the original image that are not filtered out with the initial confidence threshold (set at 50%) are expanded. It is easy to see that this confidence level results in noisy data near the edges of the object. After testing several confidence levels, a confidence acceptance level between 75% and 85% was found to be the ideal threshold.

Another important parameter that was considered was the value of the two smoothing parameters. Figure II.5 shows the effect of the smoothing parameters on

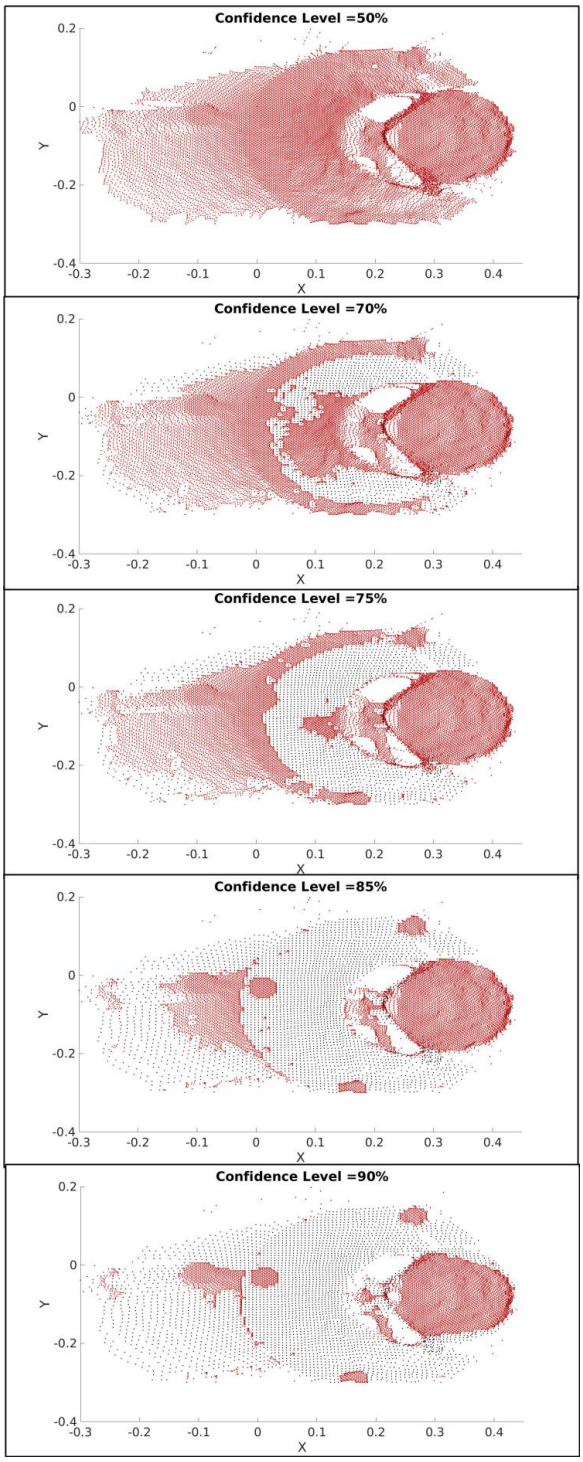

Figure II.4. Enlarged images at various confidence levels

the distribution of coordinates in physical space. Each figure is a zoomed-in view of the image projection on the X-Y plane. The black points marked with crosses are the original image points. The top-most image shows the enlarged image without any filter applied. In this figure, each point is effectively repeated several times according to the scaling factor s (2 in this case) - since no filter is applied, these points all have the same coordinate in physical space. The smoothing parameters increase moving downwards in the figure. The goal is to determine $s^2 = 2^2 = 4$ pixels and their intensities to replace each pixel in the original image. Ideally, the points in the resulting image should be fairly evenly distributed. Of these figures, images with smoothing parameters equal to 2 seem to best meet this criteria.

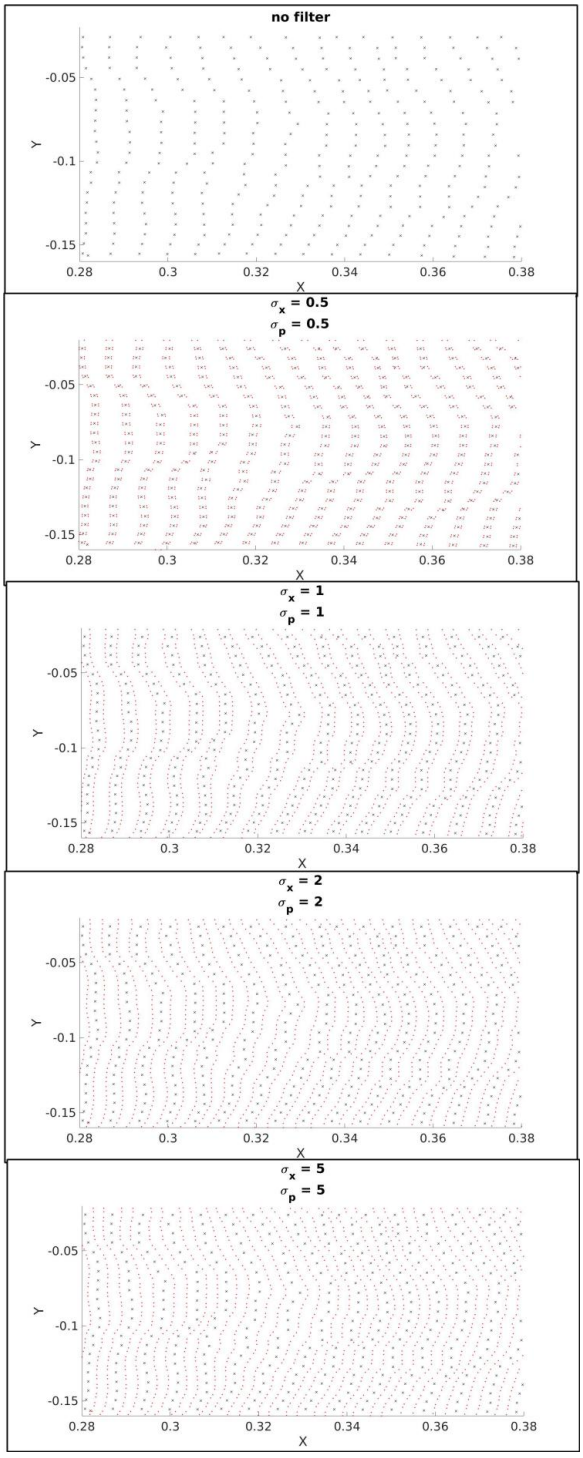

Figure II.5. Pixel intensity distribution as a function of smoothing parameters

CHAPTER III

THE ITERATIVE CLOSEST POINT ALGORITHM

The goal of the Iterative Closest Point algorithm is to minimize the difference between two point clouds. To accomplish this, a mean-squared-error metric is used. The two main error metrics used in literature are point-to-point and point-to-plane. Part of this thesis examines the difference between the two error metrics and the circumstances in which one might be preferable to the other. Some research has already been done in this area by people such as Rusinkiewicz and Levoy at Stanford University [7]. They ultimately concluded that the point-to-plane metric "performs significantly better than" the point-to-point metric. However, this result may be different with low sensor resolution, high sensor noise, or other inaccuracies such as multipath errors in LIDAR based sensors. This effect was studied in this thesis by creating rendered, "perfect" images and comparing them to actual, "imperfect" images taken with a LIDAR, by observing the ICP convergence and pose accuracy.

An overview of the system is shown in Figure III.1. If the current pose estimate is not valid, or a pose has not been obtained yet, a separate initialization module determines the pose from the camera data. Once a valid pose is constructed, a new image is taken and the pose is updated using ICP. If at anytime the pose estimate is determined to be poor, the pose is deemed invalid and is reinitialized. This thesis focuses on the part of the flowchart boxed in red. The green dashed box represents the ICP algorithm. This algorithm is expanded in greater detail in Figure III.2.

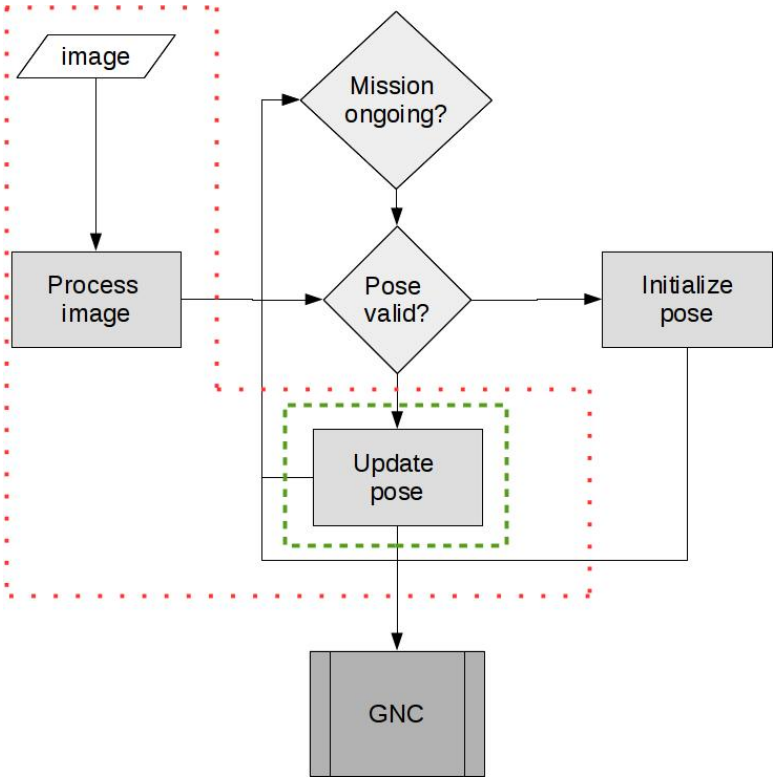

Figure III.1. Flowchart of the overall system.

In the flowchart (Figure III.2), N represents the number of iterations of ICP, and M represents the number of points in the model. It is in this algorithm that the correspondence problem becomes important. Building an accurate camera model is crucial for the success of the algorithm. For each point in the model, the algorithm predicts the corresponding sensor point by using the current best pose estimate to convert from the model frame to the sensor frame. Then, using the intrinsic parameters of the sensor, the pixel values are approximated by means of the look-up table described in II.A.2. This pixel coordinate is then used to get the actual sensor coordinate associated with that pixel. If this sensor point is good (i.e. it has not been filtered out during image processing), then the algorithm checks to see if the difference between the predicted and actual sensor points is below a predetermined, tunable

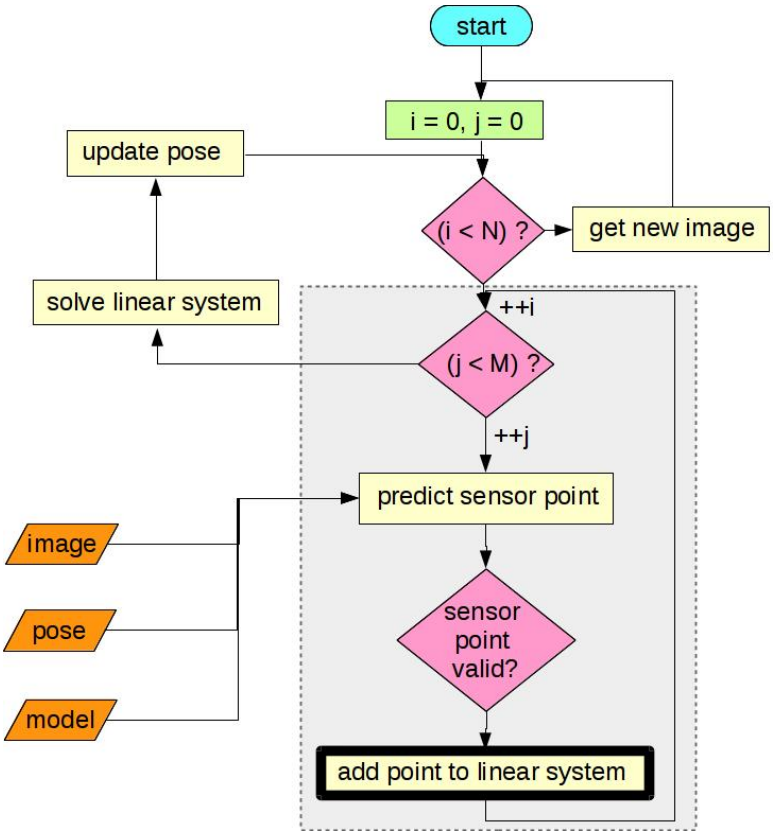

Figure III.2. Flowchart of the ICP algorithm.

threshold. It also checks to see if the normals to the surfaces of those same points are close enough together (i.e. if the dot product between the two is above a certain threshold). If all of these thresholds are met, then the point is added to the linear system as described in either III.A or III.B. This process continues until N iterations have occurred, or until ICP has been determined to have converged. Convergence is determined by a combination of multiple factors, including the number of points added to the linear system (inliers) and the pose correction norms from the current iteration.

Algorithm 6 Iterative Closest Point

function ICP($\hat{T}, model, Image$) ▷ $\hat{T}$ current pose estimate
  $\hat{R} \leftarrow \hat{T}.R$
  $\hat{t} \leftarrow \hat{T}.t$
  for each $m_i \in model$ **do**
    $s_i \leftarrow$ modelToSensor$(m_i, \hat{T})$
    $\{U, V\} \leftarrow$ XyzToPixel(s_i)
    $s \leftarrow$BestPixel$(Image, m_i, \{U, V\}, h, \hat{T})$ ▷ Find best correspondence
  end for
  if $s \neq 0$ **then**
    $q \leftarrow m_i - \hat{t}$
    $p \leftarrow \hat{R}\ s$
    $n \leftarrow ||q - p||$
    $s \leftarrow$ dot$(m_i.N, s.N)$ ▷ dot product between normal vectors
    if $s > s_0$ **and** $n < n_0$ **then**
      $w = (n - n_0)^T (n - n_0)$
      $\{\delta H, \delta y\} \leftarrow$AddToLinearSystem$(m_i, p, q, w)$
      $H \leftarrow H + \delta H$
      $y \leftarrow y + \delta y$
    end if
  end if
  $\{\delta R, \delta t\} \leftarrow$ SolveLinearSystem$(\{H, y\})$
  $\hat{T}.R \leftarrow \delta R \hat{R}$
  $\hat{T}.t \leftarrow \hat{t} + \delta t$
  return $\hat{T}$
end function

function BestPixel$(Image, m_i, \{U, V\}, h, \hat{T})$ ▷ Best point within window h
  $n_{best} \leftarrow \infty$
  $s_{best} \leftarrow 0$
  $\hat{R} \leftarrow \hat{T}.R$
  $\hat{t} \leftarrow \hat{T}.t$
  for $i = -h$ to h **do**
    for $j = -h$ to h **do**
      $s \leftarrow Image.xyz[\{U + i, V + j\}]$
      $q \leftarrow m_i - \hat{t}$
      $p \leftarrow \hat{R}\ s$
      $n \leftarrow ||q - p||$
      if $n < n_{best}$ **then**
        $n_{best} \leftarrow n$
        $s_{best} \leftarrow s$
      end if
    end for
  end for
  return s_{best}
end function

Pseudocode for the ICP algorithm is shown in Algorithm 6. Variables such as h, n_0, and s_0 are tuning parameters for the algorithm. The functions AddToLinearSystem and (to a lesser extent) SolveLinearSystem depend on the variant of ICP being used. These functions are expressed as "add point to linear system" and "solve linear system" in the ICP flowchart (Figure III.2).

III.A. Point-to-Point

In Equation 3.1, δR and $\delta \underline{r}$ are the rotation matrix and translation vector, respectively, that we desire to find.

$$\underline{y} = \delta R \underline{x} + \delta \underline{r} \tag{3.1}$$

The quantities $\underline{x}$ and $\underline{y}$ are given by Equation 3.2. $\hat{R}_{m/s}$ is the current best estimate of the rotation from the sensor to model frame, and $[\hat{r}_{s/m}]_m$ is the current best estimate of the translation from model to sensor coordinatized in the model frame. The unknowns, δR and $\delta \underline{r}$, collectively represent the differential pose between the sensor and target frames.

$$\underline{x} = \hat{R}_{m/s}[\underline{\tilde{m}}]_s \tag{3.2a}$$

$$\underline{y} = [\underline{m}]_m - [\underline{\hat{r}}_{m/s}]_m \tag{3.2b}$$

A few definitions given by Equation 3.3 will prove useful later in the derivation. In these equations, $\delta \underline{q}^X$ is the skew-symmetric matrix obtained from the Classic Rodrigues Parameters (CRPs), and I is the identity matrix.

$$\delta R = (I + \delta \underline{q}^X)^{-1}(I - \delta \underline{q}^X) \tag{3.3a}$$

$$\delta \underline{z} = (I + \delta \underline{q}^X)\delta \underline{r} \tag{3.3b}$$

$$\underline{a} = \underline{x} + \underline{y} \tag{3.3c}$$

$$\underline{b} = \underline{y} - \underline{x} \tag{3.3d}$$

Substituting Equation 3.3a into Equation 3.1 yields Equation 3.4.

$$\underline{y} = (I + \delta \underline{q}^X)^{-1}(I - \delta \underline{q}^X)\underline{x} + \delta \underline{r} \tag{3.4}$$

After multiplying both sides of the equation by $I + \delta \underline{q}^X$ and canceling terms, we are left with Equation 3.5.

$$\underline{y} + \delta \underline{q}^X \underline{y} = \underline{x} - \delta \underline{q}^X \underline{x} + \delta \underline{z} \tag{3.5}$$

Combining Equation 3.5 and Equation 3.3d yields Equation 3.6

$$\underline{b} = (\underline{x} + \underline{y})^X \delta \underline{q} + \delta \underline{z} \tag{3.6}$$

Equation 3.6 can be simplified using Equation 3.3c and can be written in matrix form, given by Equation 3.7

$$\underline{b} = \begin{pmatrix} \underline{a}^X & I \end{pmatrix} \begin{pmatrix} \delta \underline{q} \\ \\ \delta \underline{z} \end{pmatrix} \tag{3.7}$$

The error between the measured and estimated $\underline{b}$ is given by Equation 3.8. The tildes over the $\underline{a}$ and $\underline{b}$ indicate measured quantities.

$$\underline{e} = \tilde{\underline{b}} - \begin{pmatrix} \tilde{\underline{a}}^X & I \end{pmatrix} \begin{pmatrix} \delta\hat{\underline{q}} \\ \delta\hat{\underline{z}} \end{pmatrix} \tag{3.8}$$

The error we seek to minimize is the sum-squared error, given by Equation 3.9, expanded in Equation 3.10. If we take partial derivatives with respect to $\delta\hat{\underline{q}}$ and $\delta\hat{\underline{z}}$, and set them to zero, we can find the values that minimize this error. These partials are shown in Equation 3.11.

$$\min \sum_i \underline{e}_i^T \underline{e}_i \tag{3.9}$$

$$\begin{aligned} \underline{e}_i^T \underline{e}_i = {} & \tilde{\underline{b}}_i^T \tilde{\underline{b}}_i - \tilde{\underline{b}}_i^T \tilde{\underline{a}}_i^X \delta\hat{\underline{q}} - \tilde{\underline{b}}_i^T \delta\hat{\underline{z}} \\ & + \delta\hat{\underline{q}}^T \tilde{\underline{a}}_i^X \tilde{\underline{b}}_i - \delta\hat{\underline{q}}^T \tilde{\underline{a}}_i^X \tilde{\underline{a}}_i^X \delta\hat{\underline{q}} - \delta\hat{\underline{q}}^T \tilde{\underline{a}}_i^X \delta\hat{\underline{z}} \\ & - \delta\hat{\underline{z}}^T \tilde{\underline{b}}_i + \delta\hat{\underline{z}}^T \tilde{\underline{a}}_i^X \delta\hat{\underline{q}} + \delta\hat{\underline{z}}^T \delta\hat{\underline{z}} \end{aligned} \tag{3.10}$$

$$\frac{\partial(\underline{e}^T \underline{e})}{\partial(\delta\hat{\underline{q}})} = -\tilde{\underline{b}}^T \tilde{\underline{a}}^X - \tilde{\underline{b}}^T \tilde{\underline{a}}^X - \delta\hat{\underline{q}}(\tilde{\underline{a}}^X \tilde{\underline{a}}^X + \tilde{\underline{a}}^X \tilde{\underline{a}}^X) + \delta\hat{\underline{z}}^T \tilde{\underline{a}}^X + \delta\hat{\underline{z}}^T \tilde{\underline{a}}^X \tag{3.11a}$$

$$\frac{\partial(\underline{e}^T \underline{e})}{\partial(\delta\hat{\underline{z}})} = -\tilde{\underline{b}}^T - \delta\hat{\underline{q}}^T \tilde{\underline{a}}^X - \tilde{\underline{b}}^T - \delta\hat{\underline{q}}^T \tilde{\underline{a}}^X + 2\delta\hat{\underline{z}}^T \tag{3.11b}$$

Once we combine like terms and set the partials equal to zero, we obtain Equation 3.12.

$$\begin{aligned} -\tilde{\underline{b}}^T \tilde{\underline{a}}^X - \delta\hat{\underline{q}}^T \tilde{\underline{a}}^X \tilde{\underline{a}}^X + \delta\hat{\underline{z}}^T \tilde{\underline{a}}^X = 0 \\ -\tilde{\underline{b}}^T - \delta\hat{\underline{q}}^T \tilde{\underline{a}}^X + \delta\hat{\underline{z}}^T = 0 \end{aligned} \tag{3.12}$$

After taking the transpose of Equation 3.12, we obtain Equation 3.13.

$$\begin{aligned} \tilde{\underline{a}}^X \tilde{\underline{b}} - \tilde{\underline{a}}^X \tilde{\underline{a}}^X \delta\hat{\underline{q}} - \tilde{\underline{a}}^X \delta\hat{\underline{z}} = 0 \\ -\tilde{\underline{b}} + \tilde{\underline{a}}^X \delta\hat{\underline{q}} + \delta\hat{\underline{z}} = 0 \end{aligned} \tag{3.13}$$

These two equations may be written in matrix form, as shown in Equation 3.14.

$$\begin{bmatrix} \tilde{\underline{a}}^X \tilde{\underline{a}}^X & \tilde{\underline{a}}^X \\ \tilde{\underline{a}}^X & I \end{bmatrix} \begin{bmatrix} \delta\hat{\underline{q}} \\ \delta\hat{\underline{z}} \end{bmatrix} = \begin{bmatrix} \tilde{\underline{a}}^X \tilde{\underline{b}} \\ \tilde{\underline{b}} \end{bmatrix} \tag{3.14}$$

In Equation 3.14, the quantities $\delta\hat{\underline{q}}$ and $\delta\hat{\underline{z}}$ are desired. These can be obtained by performing a pseudo inverse, shown in Equation 3.15. The quantities p, H, and v are defined in Equation 3.16 and Equation 3.17.

$$p = (H^T H)^{-1} H^T v. \tag{3.15}$$

$$H_i = \begin{bmatrix} \tilde{\underline{a_i}}^X \tilde{\underline{a_i}}^X & \tilde{\underline{a_i}}^X \\ \tilde{\underline{a_i}}^X & I \end{bmatrix} \qquad p = \begin{bmatrix} \delta\hat{\underline{q}} \\ \delta\hat{\underline{z}} \end{bmatrix} \qquad v_i = \begin{bmatrix} \tilde{\underline{a_i}}^X \tilde{\underline{b_i}} \\ \tilde{\underline{b_i}} \end{bmatrix} \tag{3.16}$$

$$H = \sum_i w_i \, H_i \qquad v = \sum_i w_i \, v_i \tag{3.17}$$

$$w_i = (\underline{y}_i - \underline{x}_i)^T (\underline{y}_i - \underline{x}_i) \tag{3.18}$$

Once $\delta\hat{\underline{q}}$ and $\delta\hat{\underline{z}}$ are found, the pose (δR and $\delta\underline{r}$) can be obtained from using Equation 3.3a and Equation 3.3b. Note the rank deficiency of H_i. This may seem problematic; however, this matrix is never actually inverted (pseudo- or otherwise). The H matrix, whose rank does matter, represents a weighted summation over all points in the model. These weights are defined in Equation 3.18. Taken collectively, this matrix will in general be non-singular. This matrix approaches a singularity as the number of points in the linear system approaches 1. For this reason, a pseudoinverse is used to invert H in case there is a small number of inliers. Pseudocode for the point-to-point ICP variant is given in Algorithm 7 and Algorithm 8.

Algorithm 7 Build Linear System, Point-to-Point

function AddToLinearSystem(m_i, p, q, w)

$A \leftarrow \text{Skew}(q + p)$ ▷ Skew Symmetric Matrix $A = (q + p)^X$

$b \leftarrow (q - p)$

$H_{11} \leftarrow A^2$

$H_{12} \leftarrow A$

$H_{21} \leftarrow A$

$H_{22} \leftarrow I$

$y_1 \leftarrow A\ b$

$y_2 \leftarrow b$

$$H \leftarrow w \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$$

$$y \leftarrow w \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

return $\{H, y\}$

end function

Algorithm 8 Solve Linear System, Point-to-Point

function SolveLinearSystem($\{H, y\}$)

$\delta v \leftarrow (H^T H)^{-1} H^T y$

$\{\delta q, \delta z\} \leftarrow \delta v$

$\delta Q \leftarrow \text{Skew}(\delta q)$

$\delta R \leftarrow (I + \delta Q)^{-1}(I - \delta Q)$

$\delta t \leftarrow (I + \delta Q)^{-1} \delta z$

return $\{\delta R, \delta t\}$

end function

III.B. Point-to-Plane

The point-to-plane follows a very similar derivation to the point-to-point method. In this case, we are minimizing the error in the normal direction to the model points (in the sensor's frame). This error is given by Equation 3.19. This is the error between the estimated $\underline{\hat{r}}$ and the transformed measurement $\underline{\tilde{r}}$. These terms are equivalent to the terms $\underline{y}$ and $\underline{x}$, respectively, from the point-to-point derivation.

$$\underline{e} = N^T(\underline{\hat{r}} - (\delta R\underline{\tilde{r}} + \delta\underline{r})) \tag{3.19}$$

Since the attitude correction in each step of ICP will be very small, we can linearize δR by defining it as in Equation 3.20. This will make solving for the attitude will be much simpler.

$$\delta R = I - \delta\alpha^X \tag{3.20}$$

Substituting Equation 3.20 into Equation 3.19 yields Equation 3.21

$$\underline{e} = N^T\underline{\hat{r}} - N^T\underline{\tilde{r}} + N^T\underline{\tilde{r}}^X\delta\underline{\alpha} - N^T\delta\underline{r} \tag{3.21}$$

Like in the point-to-point method, the error we seek to minimize is given by $\underline{e}^T\underline{e}$, shown in Equation 3.22.

$$\begin{aligned}
\underline{e}^T\underline{e} = {} & \underline{\hat{r}}^T NN^T\underline{\hat{r}} - \underline{\hat{r}}^T NN^T\underline{\tilde{r}} + \underline{\hat{r}}^T NN^T\underline{\tilde{r}}^X\delta\underline{\alpha} - \underline{\hat{r}}^T NN^T\delta\underline{r} \\
& - \underline{\tilde{r}}^T NN^T\underline{\hat{r}} + \underline{\tilde{r}}^T NN^T\underline{\tilde{r}} - \underline{\tilde{r}}^T NN^T\underline{\tilde{r}}^X\delta\underline{\alpha} + \underline{\tilde{r}}^T NN^T\delta\underline{r} \\
& - \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T\underline{\hat{r}} + \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T\underline{\tilde{r}} - \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T\underline{\tilde{r}}^X\delta\underline{\alpha} + \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T\delta\underline{r} \\
& - \delta\underline{r}^T NN^T\underline{\hat{r}} + \delta\underline{r}^T NN^T\underline{\tilde{r}} - \delta\underline{r}^T NN^T\underline{\tilde{r}}^X\delta\underline{\alpha} + \delta\underline{r}^T NN^T\delta\underline{r}
\end{aligned} \tag{3.22}$$

Again, taking the partial derivative with respect to $\delta\underline{\alpha}$ and $\delta\underline{r}$, we obtain Equation 3.23.

$$\begin{aligned}\frac{\partial(\underline{e}^T\underline{e})}{\partial(\delta\underline{r})} &= -\underline{\hat{r}}^T NN^T + \underline{\tilde{r}}^T NN^T + \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T - \underline{\hat{r}}^T NN^T \\ &\quad + \underline{\tilde{r}}^T NN^T + \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T + 2\delta\underline{r}^T NN^T \end{aligned} \tag{3.23a}$$

$$\begin{aligned}\frac{\partial(\underline{e}^T\underline{e})}{\partial(\delta\underline{\alpha})} &= \underline{\hat{r}}^T NN^T\underline{\tilde{r}}^X - \underline{\tilde{r}}^T NN^T\underline{\tilde{r}}^X + \underline{\hat{r}}^T NN^T\underline{\tilde{r}}^X - \underline{\tilde{r}}^T NN^T\underline{\tilde{r}}^X \\ &\quad - 2\delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T\underline{\tilde{r}}^X - \delta\underline{r}^T\underline{\tilde{r}}^X NN^T\underline{\tilde{r}}^X - \delta\underline{r}^T NN^T\underline{\tilde{r}}^X \end{aligned} \tag{3.23b}$$

After collecting terms and setting the partial derivatives equal to zero, we obtain Equation 3.24

$$\begin{aligned} -\underline{\hat{r}}^T NN^T + \underline{\tilde{r}}^T NN^T + \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T + \delta\underline{r}^T NN^T = 0 \\ \underline{\hat{r}}^T NN^T\underline{\tilde{r}}^X - \underline{\tilde{r}}^T NN^T\underline{\tilde{r}}^X - \delta\underline{\alpha}^T\underline{\tilde{r}}^X NN^T\underline{\tilde{r}}^X - \delta\underline{r}^T NN^T\underline{\tilde{r}}^X = 0 \end{aligned} \tag{3.24}$$

Taking the transpose of Equation 3.24, we obtain Equation 3.25, which can be rearranged in matrix form as shown in Equation 3.26.

$$\begin{aligned} -NN^T\underline{\hat{r}} + NN^T\underline{\tilde{r}} - NN^T\underline{\tilde{r}}^X\delta\underline{\alpha} + NN^T\delta\underline{r} = 0 \\ -\underline{\tilde{r}}^X NN^T\underline{\hat{r}} + \underline{\tilde{r}}^X NN^T\underline{\tilde{r}} - \underline{\tilde{r}}^X NN^T\underline{\tilde{r}}^X\delta\underline{\alpha} + \underline{\tilde{r}}^X NN^T\delta\underline{r} = 0 \end{aligned} \tag{3.25}$$

$$\begin{bmatrix} NN^T & -NN^T\underline{\tilde{r}}^X \\ \underline{\tilde{r}}^X NN^T & -\underline{\tilde{r}}^X NN^T\underline{\tilde{r}}^X \end{bmatrix} \begin{bmatrix} \delta\underline{r} \\ \delta\underline{\alpha} \end{bmatrix} = \begin{bmatrix} NN^T(\underline{\hat{r}} - \underline{\tilde{r}}) \\ \underline{\tilde{r}}^X NN^T(\underline{\hat{r}} - \underline{\tilde{r}}) \end{bmatrix} \tag{3.26}$$

This equation can be solved by means of a pseudoinverse, as shown in Equation 3.15. The quantities p, H_i, and v_i are defined in Equation 3.27. Equations 3.17 and 3.18 still apply to the point-to-plane formulation. As in the point-to-point variant, this variant also has a singular H_i matrix - the same justification applies as in the previous section.

$$H_i = \begin{bmatrix} N_i N_i^T & -N_i N_i^T \tilde{\underline{r}}_i^X \\ \tilde{\underline{r}}_i^X N_i N_i^T & -\tilde{\underline{r}}_i^X N_i N_i^T \tilde{\underline{r}}_i^X \end{bmatrix} \qquad p = \begin{bmatrix} \delta \underline{r} \\ \delta \underline{\alpha} \end{bmatrix} \qquad v_i = \begin{bmatrix} N_i N_i^T (\hat{\underline{r}}_i - \tilde{\underline{r}}_i) \\ \tilde{\underline{r}}_i^X N_i N_i^T (\hat{\underline{r}}_i - \tilde{\underline{r}}_i) \end{bmatrix} \tag{3.27}$$

Finally, Equation 3.20 can be used to find δR from $\delta \underline{\alpha}$. Pseudocode for the point-to-point variant is given in Algorithm 9 and Algorithm 10.

Algorithm 9 Build Linear System, Point-to-Plane

function AddToLinearSystem(m_i, p, q, w)
  $n \leftarrow m_i.N$ ▷ normal vector for model point m_i
  $P \leftarrow \text{Skew}(p)$
  $H_{11} \leftarrow nn^T$
  $H_{12} \leftarrow -nn^T P$
  $H_{21} \leftarrow Pnn^T$
  $H_{22} \leftarrow -Pnn^T P$
  $y_1 \leftarrow nn^T(q - p)$
  $y_2 \leftarrow Pnn^T(q - p)$
  $H \leftarrow w \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$
  $y \leftarrow w \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$
  return $\{H, y\}$
end function

Algorithm 10 Solve Linear System, Point-to-Plane

function SolveLinearSystem($\{H, y\}$)
  $\delta v \leftarrow (H^T H)^{-1} H^T y$
  $\{\delta r, \delta \alpha\} \leftarrow \delta v$
  $\delta A \leftarrow \text{Skew}(\delta \alpha)$
  $\delta R \leftarrow (I - \delta A)$
  $\delta t \leftarrow \delta r$
  return $\{\delta R, \delta t\}$
end function

III.C. Parallelization of ICP

At first, it seems counter-intuitive to be able to parallelize an algorithm with the name "iterative" in its title — "iterative" implies that one iteration depends on the other, therefore making it impossible to parallelize. However, it is in fact possible to parallelize the ICP algorithm. Essentially, during the building of the linear system (part of the boxed process in Figure III.2), the model (CAD file, etc) is split into multiple parts and assigned to different threads. This is possible because the model points are independent of one another in the building of the linear system. Once all threads have finished, the linear system is solved and the next ICP iteration may begin. This process is shown in Figure III.3. Note the similarity to the unparallelized version in Figure III.2. Instead of iterating over the model points sequentially, the model points are divided into subsections that can be processed in parallel. The values of p and n in the flowchart are the lower and upper indices of the points in the model. In the figure, m is the total number of threads, M is the total number of points in the model, and k is the thread number currently being processed.

The ICP algorithm was run several times, each using a different number of threads (from 1 to 15). It was determined that 7 threads was the optimal number of threads to use. The machine that this was implemented on has a 4-core Intel-i7 processor. Figure III.4 shows the average time for one iteration of ICP as a function of the number of threads. Since the computer had a 4-core processor, the jumps at multiples of 4 make sense — In fact, the run time for 12 threads is off the chart. Once the number of threads goes beyond 7 the addition of more threads is no longer advantageous.

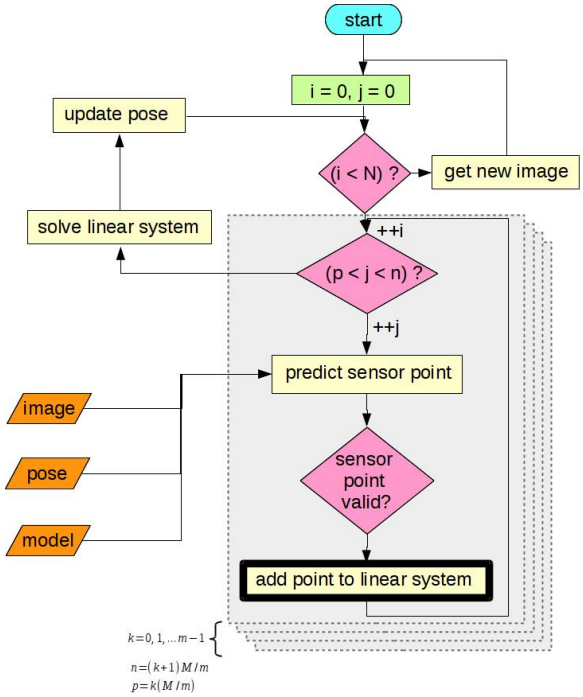

Figure III.3. Parallelization of ICP.

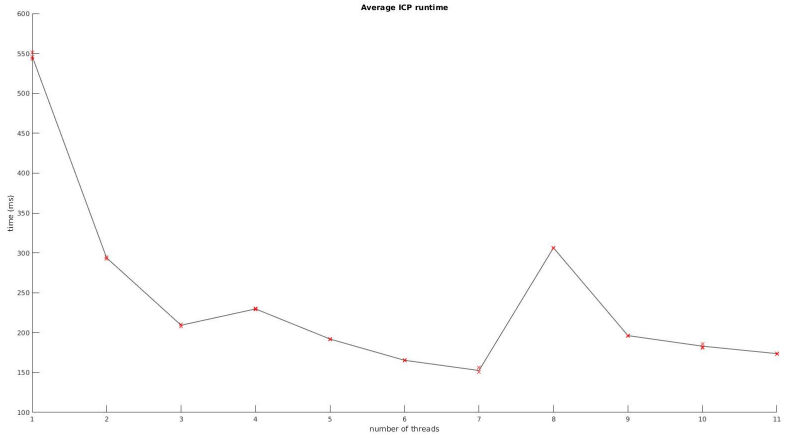

Figure III.4. Parallelization runtimes.

III.D. The Problem of Pose Initialization

Even though this thesis does not focus on the pose initialization aspect of the pose estimation problem, it is still an important consideration to the overall estimation algorithm and should not be overlooked. The Iterative Closest Point algorithm requires a current estimate of the pose in order to provide an updated estimate. The current estimate is used as an initial guess for the point cloud alignment. ICP then performs iterative optimization to compute a correction to the prior estimate. Without a valid initial guess, the algorithm will fail to converge to the true global optimum. ICP only works for very small corrections - if the initial point cloud alignment is off by too large an amount, ICP might converge to a local minima or may not converge at all. Because no pose estimate is available at system start up, an initial guess must be generated by alternative means. Furthermore, if at any time the ICP algorithm diverges, the pose must be reinitialized. A separate pose initialization module was written for this purpose.

The pose initialization module is based on a method by Drost [8]. The main idea is as follows — consider a point-pair correspondence: two points and their surface normals in the sensor point cloud correspond to two points and their surface normals in the model point cloud. The transformation from the local coordinates (denoted by subscript m) to the scene coordinates (denoted by subscript s) is given by Equation 3.28.

$$s_i = T_{s\to g}^{-1} R_x(\alpha) T_{m\to g} m_i \tag{3.28}$$

If such a correspondence is known, the relative pose between the sensor point cloud the model point cloud can be uniquely determined (except in some degenerate configurations). The initialization algorithm determines a very large number of potential point-pair correspondences. Each correspondence votes for a particular pose hypothesis in a manner similar to the Generalized Hough Transform [9]. Similar pose hypotheses are then clustered together based on the user-defined threshold and the hypothesis with the most votes is selected as the best. See Figure III.5 for a summary of the Drost method.

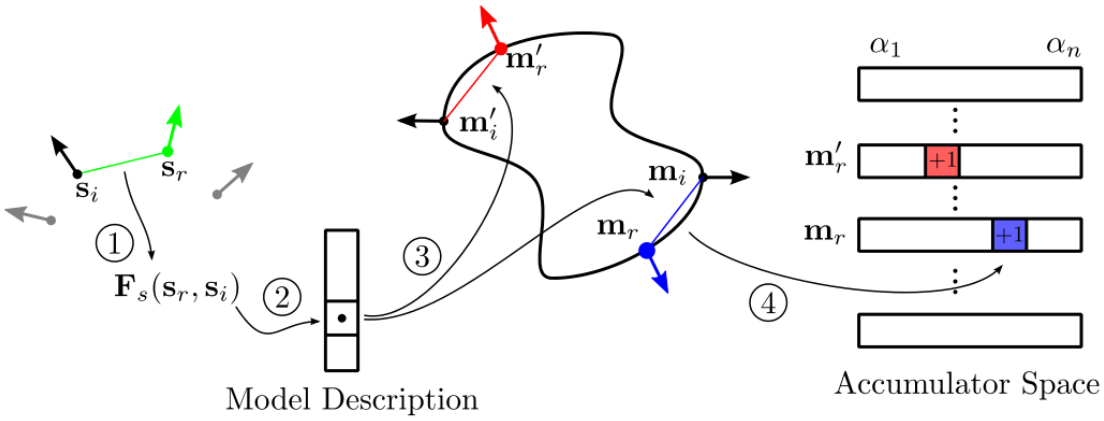

Figure III.5. Drost voting scheme: Graphic by Drost [8].

CHAPTER IV

RESULTS

Both variants of the ICP algorithm were tested given three other variables. These variables include: 1) image size, 2) ICP iterations, and 3) noise level. These tests were performed on image sets of different initial conditions and spin rates. Two different image sizes were used - the original camera image size (176x144) and images that had been processed as in Subsection II.B.3 with a scaling factor of 2. Two different cases of the ICP iteration variable were tested. The first is a fixed number of iterations for all images. In the second case, the algorithm stops iterating if the correction norm of the pose is below a certain threshold, but will not exceed the number of iterations from the first case. For the noise level variable, simulated images with no noise (Set 1 - Set 3), simulated images with added noise (Set 1 - Set 3), and real images collected by the sensor were used (Set 0 only). The parameters of each image set are listed below.

- Set 0:

  Real images from sensor

  Initial 3-1-2 Euler angles = [1.40, 3.63, -159] deg

  Yaw rate = 1.87 deg/frame

- Set 1:

  Simulated images

  Initial 3-1-2 Euler angles = [165.6, -12.73, -130.7] deg

  Yaw rate = 1.87 deg/frame

- Set 2:

  Simulated images

  Initial 3-1-2 Euler angles = [-175.4, 10.98, 22.95] deg

  Yaw rate = 2.08 deg/frame

- Set 3:

  Simulated images

  Initial 3-1-2 Euler angles = [-177.2, 6.34, 24.10] deg

  Yaw rate = 2.18 deg/frame

Figure IV.1 shows the average error of each simulated set for combinations of small images, large images, point-to-point metric, and point-to-plane metric. The four data points for each average are the 4 combinations of the number of ICP iterations and noise level.

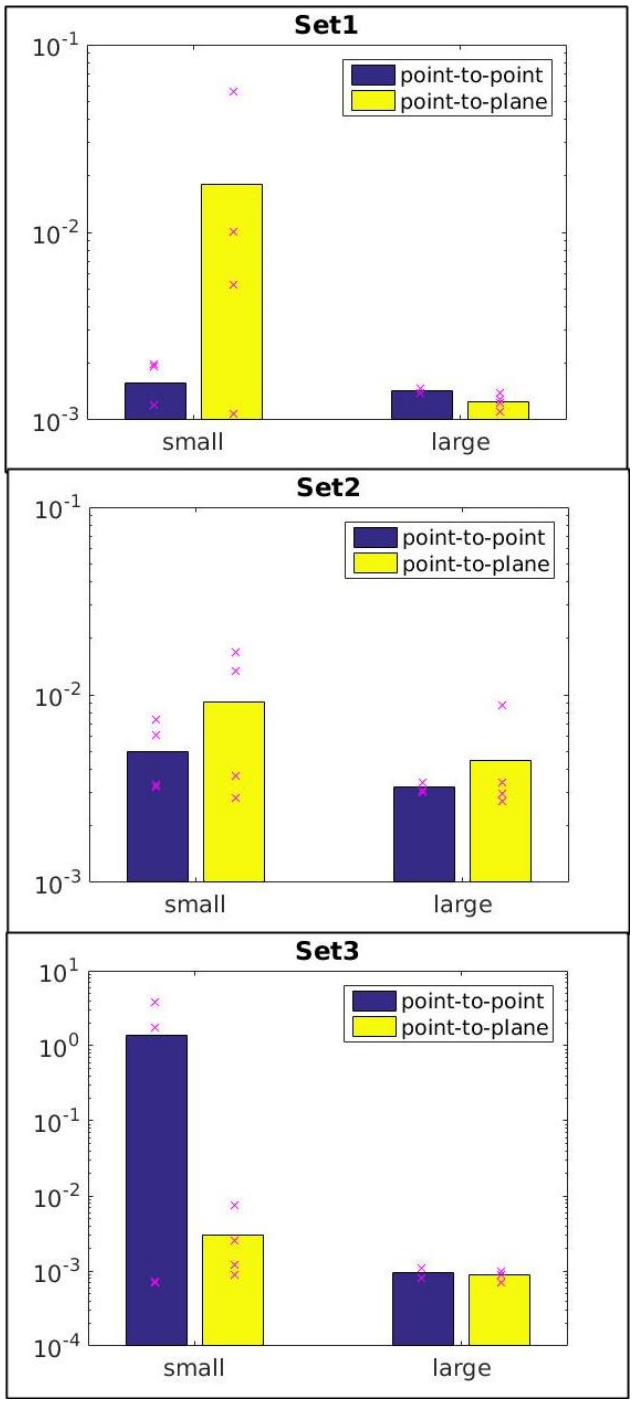

Figure IV.1. Errors for simulated image sets

There is a distinct pattern in the simulated data set results. In every case tested, using the method presented in this thesis to enlarge the images decreases the average error. Furthermore, using the smaller images resulted in a larger spread of error across different parameters for both ICP metrics. Overall, the point-to-plane metric with the large images produced the best result. Similar results were found with the real set of images (see Figure IV.2).

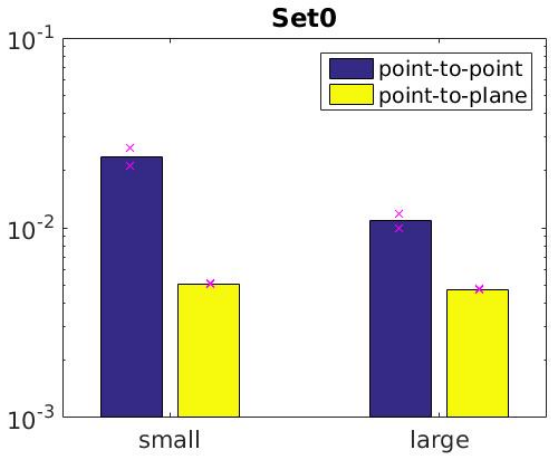

Figure IV.2. Errors for real image sets

For the real camera image set, the pose estimates and errors as a function of frame number for the best case tested (large image, point-to-plane) are shown in Figure IV.3. The worst case (small image, point-to-point) is shown in Figure IV.4. Clearly, the former case produced cleaner estimates and had a smaller error overall.

Importantly, the runtimes for all test cases were measured. Fortunately, the runtime of the ICP algorithm itself does not intrinsically depend on the size of the

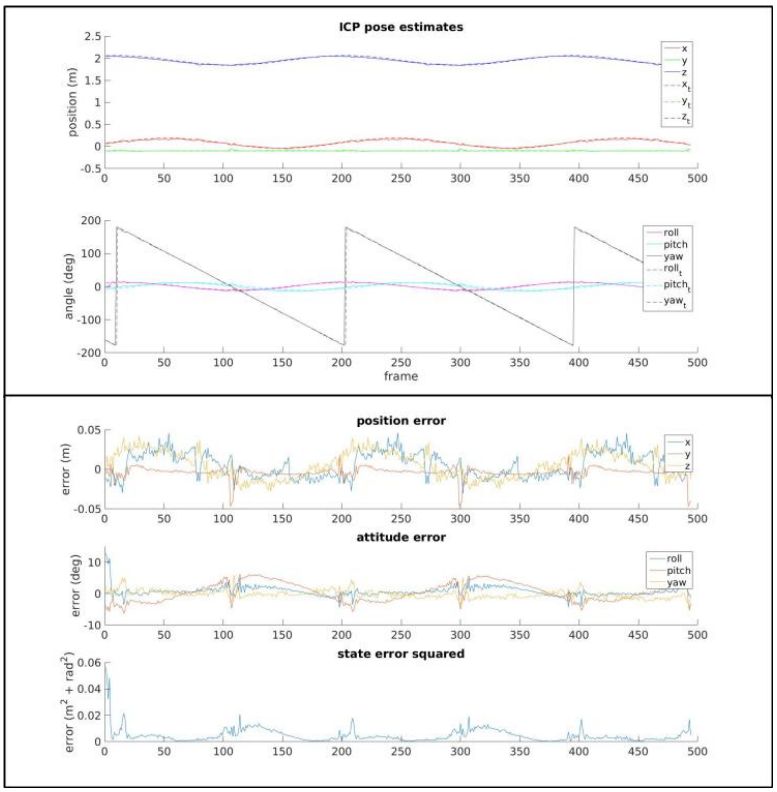

Figure IV.3. Best case pose estimates and errors.

image. This is because it iterates over the points in the model, rather than the image. There are two primary ways that the image size affects the overall runtime of the program. The first is the initial processing of the image. This doesn't take much longer because the bilateral filtering and expanding of the image can be done at the same time as the bilateral filter that gets applied anyway (to compute the image normals). The other slow-down occurs as an artifact of passing slightly larger amounts of data to and from various parts of the program. However, this does have a measurable effect on the runtime, as shown in Figure IV.5.

The difference in runtimes between the small and large images is about 20 milliseconds, or about a 8% increase in runtime. It should be noted that the plot's y-axis does not start at zero — this was done in order to more easily see the individual run-

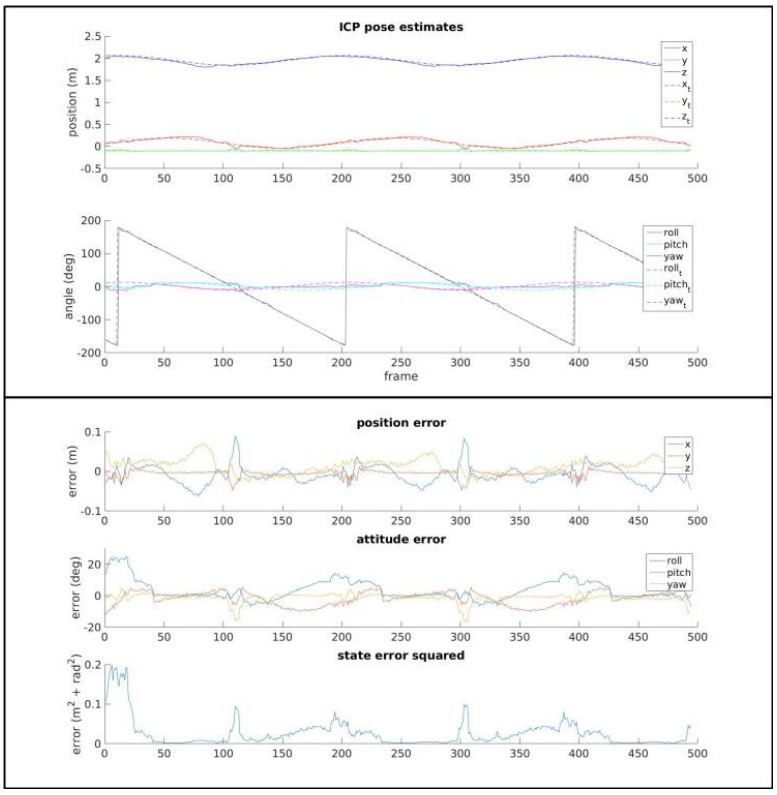

Figure IV.4. Worst case pose estimates and errors.

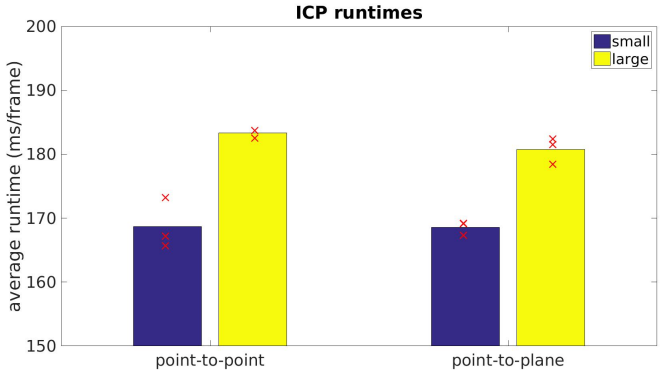

Figure IV.5. Runtimes for enlarged images

times and the differences between the average runtimes. The difference in runtimes could be decreased by parallelizing the image processing module of the program. For

instance, in the bilateral filter, the updated value at each pixel can be determined independently from the other pixels. Thus, parallelization can be easily done. This should greatly reduce the gap between the computation times. The best method to select depends on the computational abilities of the machine and on the level of noise expected based on environmental factors such as ambient lighting, background, and sensor interference.

CHAPTER V

KALMAN FILTER

In some cases, the ICP algorithm may provide noisy estimates due to sensor limitations. These may include problems such as multi-path errors, low resolution, and small field-of-view. A Kalman filter has been implemented to mitigate these issues. Furthermore, the vision measurements produced by ICP only reflect relative motion between the sensor and the object. In order to provide inertial navigation, an IMU is added to the system. This IMU provides inertial acceleration and attitude measurements of the capture vehicle to which the sensor is attached. The relative position and attitude between the sensor and IMU frames remains fixed, as it is assumed that the IMU is fixed to the sensor apparatus.

V.A. State and Measurement Definitions

In this implementation, the states that are tracked are given by Equation 5.1.

$$\underline{x} = \begin{bmatrix} [\underline{t}_{m/s}]_n \\ [\underline{v}_{m/s}]_n \\ \underline{\dot{\alpha}}_{m/s} \\ \underline{\alpha}_{m/s} \\ \underline{\omega} \end{bmatrix} \tag{5.1}$$

The vector $[\underline{t}_{m/s}]_n$ is the position of the model with respect to the sensor coordinatized in the inertial frame. The vector $[\underline{v}_{m/s}]_n$ is the inertial derivative of this position. The vector $\underline{\alpha}_{m/s}$ is the 3-1-2 Euler rotation sequence from the sensor frame to the model frame. The derivative of these angles is given by $\underline{\dot{\alpha}}_{m/s}$. The body-fixed

angular velocity is given by $\underline{\omega}$.

The measurements provided from the Iterative Closest Point algorithm are given by Equation 5.2.

$$\underline{\tilde{y}} = \begin{bmatrix} [\underline{\tilde{t}}_{m/s}]_s \\ \\ \underline{\tilde{\alpha}}_{m/s} \end{bmatrix} \tag{5.2}$$

In addition, the IMU provides the acceleration of the IMU with respect to the inertial frame coordinatized in the IMU frame (denoted by $[\underline{a}_{u/n}]_u$), as well as the attitude of the IMU with respect to the inertial frame (denoted by $\underline{R}_{u/n}$).

The following two sections provide the equations that describe the Kalman filter used in this system. This formulation follows closely to that of Junkins' [10].

V.B. Propagation

In this section, a linear system model is derived and the propagation equations are determined. The model of the system including process noise is of the form given by Equation 5.3. The filter propagation equations will be of the same form.

$$\underline{x}_{k+1} = \Phi_k \underline{x}_k + \Gamma \underline{u}_k + Y \underline{w}_k \tag{5.3}$$

The process noise $\underline{w}_k$ is assumed to be Gaussian with zero mean and covariance Q_k. In this formulation, the matrix Y is assumed to be identity for simplicity.

For this system, two major assumptions are made: constant angular velocity ($\dot{\underline{\omega}} = 0$) and constant translational velocity ($[\underline{a}_{m/n}]_n = 0$). For the application of tracking objects in space, this is a reasonable assumption since the target is assumed to be free from external forces that would cause it to accelerate in any degree of freedom.

To derive the filter propagation equations, we first need to obtain the acceleration of the target with respect to the sensor coordinatized in the inertial frame, $[\underline{a}_{m/s}]_n$. This is given by Equation 5.4.

$$[\underline{a}_{m/s}]_n = [\underline{a}_{m/n}]_n + [\underline{a}_{n/u}]_n + [\underline{a}_{u/s}]_n \tag{5.4}$$

Since the IMU is fixed relative to the sensor, the term $[\underline{a}_{u/s}]_n$ is zero. We have already established that since the target is not accelerating with respect to the inertial frame, the term $[\underline{a}_{m/n}]_n$ is also zero. With these two simplifications, Equation 5.4 reduces to

$$[\underline{a}_{m/s}]_n = [\underline{a}_{n/u}]_n = -[\underline{a}_{u/n}]_n \tag{5.5}$$

The IMU measurement includes the acceleration due to gravity, so this must be subtracted out from the IMU measurement. Since the IMU measurements are known in terms of $[\underline{a}_{u/n}]_u$ and $\underline{R}_{u/n}$, we can write the IMU acceleration in terms of known quantities as

$$[\underline{a}_{u/n}]_n = \underline{R}^T_{u/n}[\underline{a}_{u/n}]_u - g \tag{5.6}$$

Combining this with Equation 5.5, the acceleration of the target with respect to the sensor coordinatized in the inertial frame is given by Equation 5.7.

$$[\underline{a}_{m/s}]_n = g - \underline{R}^T_{u/n}[\underline{a}_{u/n}]_u \tag{5.7}$$

Now that $[\underline{a}_{m/s}]_n$ is known, the translational position and velocity can be determined. The discrete kinematic equations for these quantities are shown in Equation 5.8.

$$([\hat{\underline{v}}_{m/s}]_n)_{k+1}{}^{-} = ([\hat{\underline{v}}_{m/s}]_n)_k{}^{+} + \delta t(g - (\underline{R}_{u/n})_k^T([\underline{a}_{u/n}]_u)_k) \tag{5.8a}$$

$$([\hat{\underline{t}}_{m/s}]_n)_{k+1}{}^{-} = ([\hat{\underline{t}}_{m/s}]_n)_k{}^{+} + \delta t([\hat{\underline{v}}_{m/s}]_n)_k{}^{+} + \frac{1}{2}\delta t^2(g - (\underline{R}_{u/n})_k^T([\underline{a}_{u/n}]_u)_k) \tag{5.8b}$$

Because the body-fixed angular velocity is assumed to be constant, the propagation equation for $\underline{\omega}$ is straightforward and given by Equation 5.11b. The propagation equation for $\underline{\alpha}_{m/s}$ is also straightforward and given in Equation 5.11a. However, the fact that the body-fixed angular rate is constant does not imply that the time rate of change of the Euler angles is constant. The two angular rates are only equivalent for infinitesimal Euler angles. The quantities $\dot{\underline{\alpha}}$ and $\underline{\omega}$ are related through the values of the Euler angles, as outlined in [11]. The body-fixed angular velocity $\{\underline{w}_1, \underline{w}_2, \underline{w}_3\}$ given a 3-1-2 Euler rotation sequence $\alpha = \{\underline{\alpha}_3, \underline{\alpha}_1, \underline{\alpha}_2\}$ is given by

$$\underline{\omega} = \begin{bmatrix} -\cos\alpha_1 \sin\alpha_2 & \cos\alpha_2 & 0 \\ \sin\alpha_1 & 0 & 1 \\ \cos\alpha_1 \cos\alpha_2 & \sin\alpha_2 & 0 \end{bmatrix} \dot{\underline{\alpha}} \tag{5.9}$$

Inverting this equation will allow us to determine the time rate of change of the Euler angles in terms of the body-fixed angular rate, as shown in Equation 5.10.

$$\dot{\underline{\alpha}} = \begin{bmatrix} -\sin\alpha_2/\cos\alpha_1 & 0 & \cos\alpha_2/\cos\alpha_1 \\ \cos\alpha_2 & 0 & \sin\alpha_2 \\ \tan\alpha_1 \sin\alpha_2 & 1 & -\tan\alpha_1 \cos\alpha_2 \end{bmatrix} \underline{\omega} \equiv B\underline{\omega} \tag{5.10}$$

All propagation equations involving the attitude of the target are summarized in Equation 5.11.

$$(\hat{\underline{\alpha}}_{m/s})_{k+1}{}^{-} = (\hat{\underline{\alpha}}_{m/s})_k{}^{+} + \delta t(\hat{\dot{\underline{\alpha}}}_{m/s})_k{}^{+} \tag{5.11a}$$

$$\hat{\underline{\omega}}_{k+1}{}^{-} = \hat{\underline{\omega}}_k{}^{+} \tag{5.11b}$$

$$\hat{\dot{\underline{\alpha}}}_{k+1}{}^{-} = B\hat{\underline{\omega}}_k{}^{+} \tag{5.11c}$$

Equations 5.8 and 5.11 can be written in a compact matrix form, given by Equation 5.12a. The covariance of the state estimate is also propagated as in Equation 5.12b.

$$\hat{\underline{x}}_{k+1}{}^{-} = \Phi_k \hat{\underline{x}}_k{}^{+} + \Gamma \underline{u}_k \tag{5.12a}$$

$$P_{k+1}{}^{-} = \Phi_k P_k{}^{+} \Phi_k^T + Y_k Q_k Y_k{}^T \tag{5.12b}$$

The terms $\hat{\underline{x}}_k$, $\underline{u}_k$, and Φ_k are defined in Equations 5.13, 5.14, and 5.15, respectively. Γ is simply defined as the 15×15 identity matrix. The matrix Y is the same as that of Equation 5.3. In Equation 5.15, the I and 0 terms are of dimension 3×3, and Δt is simply the propagation time step (δt) times I.

$$\hat{\underline{x}}_k = \begin{bmatrix} ([\hat{\underline{t}}_{m/s}]_n)_k \\ ([\hat{\underline{v}}_{m/s}]_n)_k \\ (\hat{\dot{\underline{\alpha}}}_{m/s})_k \\ (\hat{\underline{\alpha}}_{m/s})_k \\ (\hat{\underline{\omega}})_k \end{bmatrix} \tag{5.13}$$

$$\underline{u}_k = \begin{bmatrix} \delta t(g - (\underline{R}_{u/n})_k^T([\underline{a}_{u/n}]_u)_k) \\ \frac{1}{2}\delta t^2(g - (\underline{R}_{u/n})_k^T([\underline{a}_{u/n}]_u)_k) \\ 0_{3\times 1} \\ 0_{3\times 1} \\ 0_{3\times 1} \end{bmatrix} \tag{5.14}$$

$$\Phi_k = \begin{bmatrix} I & \Delta t & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & B_k \\ 0 & 0 & \Delta t & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \tag{5.15}$$

V.C. Update

Recall the manner in which the measurements are defined in Equation 5.2. The states of the filter and the measurements are of different dimension and also require a reference frame transformation. The measurement model for this system is given by Equation 5.16.

$$\underline{\tilde{y}}_k = H_k \underline{x}_k + \underline{\nu}_k \tag{5.16}$$

The measurement noise $\underline{\nu}_k$ is assumed to be Gaussian with zero mean and covariance R_k. This measurement covariance is a function of the pose correction norms and the number of inliers from the ICP algorithm. The measurement model is expanded in Equation 5.17.

$$\begin{bmatrix} ([\tilde{\underline{t}}_{m/s}]_s)_k \\ (\tilde{\underline{\alpha}}_{m/s})_k \end{bmatrix} = \begin{bmatrix} (R_{s/n})_k & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 \end{bmatrix} \begin{bmatrix} ([\underline{t}_{m/s}]_n)_k \\ ([\underline{v}_{m/s}]_n)_k \\ (\underline{\dot{\alpha}}_{m/s})_k \\ (\underline{\alpha}_{m/s})_k \\ (\underline{\omega})_k \end{bmatrix} + \begin{bmatrix} (\underline{\nu}_t)_k \\ (\underline{\nu}_\alpha)_k \end{bmatrix} \tag{5.17}$$

The state $\underline{\hat{x}}_k$ and its covariance P_k is updated by means of Equation 5.18.

$$\underline{\hat{x}}_k{}^+ = \underline{\hat{x}}_k{}^- + K_k(\underline{\tilde{y}}_k - H_k\underline{\hat{x}}_k{}^-) \tag{5.18a}$$

$$P_k{}^+ = (I - K_k H_k)P_k{}^- \tag{5.18b}$$

The Kalman gain, K_k is defined in Equation 5.19.

$$K_k = P_k{}^- H_k{}^T S_k{}^{-1} \tag{5.19}$$

The term S_k is called the innovation covariance and is defined in Equation 5.20

$$S_k = H_k P_k H_k{}^T + R \tag{5.20}$$

V.D. Results

The filter implemented in this thesis was first tested using simulated pose measurements so that the ICP algorithm and the filter could be tested independently of one another. The results presented in this section simulate an object approximately 2 meters away from the sensor which is stationary relative to the sensor translationally. The object is rotating at a constant rate about the body-fixed y-axis at 2 radians per frame. The filter uses a propagation time of 0.1 time units. Figure V.1 shows

the tracking of all 6 degrees of freedom of the object. The solid lines in the figures represent the pose measurement, and the dashed lines represent the filter estimates. The darker lines are the 2-sigma bounds obtained from the state covariance.

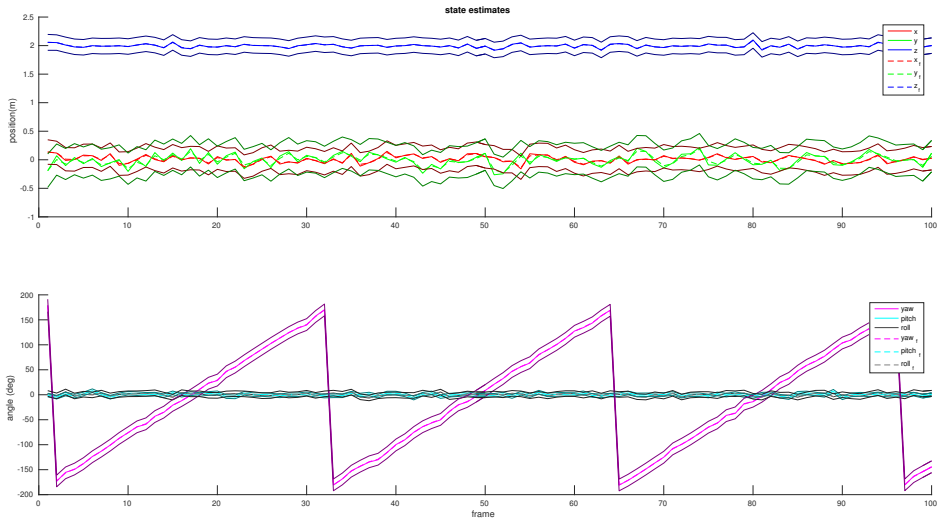

Figure V.1. Kalman filter preliminary results.

The error between the true state (which is known exactly since this is done through simulation) and the state estimate are shown in Figure V.2. These errors are on average relatively low. The figure also includes the innovation and innovation covariance (2-sigma) at each time step. The innovations are zero mean and are within the 2-sigma bounds, as they should be.

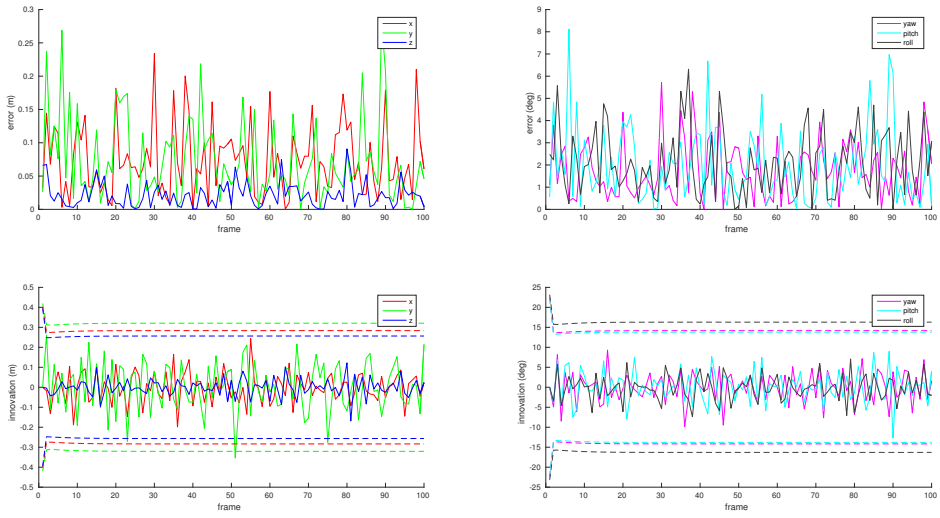

Figure V.2. Error and innovations.

Statistical tests were performed using the NEES (normalized estimation error squared) metric [12]. In a consistent filter, the normalized error squared is a chi-squared distribution such that the expected value of this error is equal to the dimension of the state from which the error is calculated. This result is derived in the following way: Let the normalized squared error p at time step k be given by Equation 5.21

$$\underline{p}(k) = (\underline{x}(k) - \underline{\hat{x}}(k|k))^T P^{-1}(k|k)(\underline{x}(k) - \underline{\hat{x}}(k|k)) \tag{5.21}$$

This equation can be transformed into the form of Equation 5.22 using the substitution given in Equation 5.23

$$\underline{p}(k) = \underline{y}(k)^T \underline{y}(k) \tag{5.22}$$

$$\underline{y}(k) = P^{-1/2}(k|k)(\underline{x}(k) - \underline{\hat{x}}(k|k)) \tag{5.23}$$

The variable $\underline{y}$ is normally distributed with zero mean and identity variance. It follows that the quantity $\underline{y}^T \underline{y}$ (and therefore $\underline{p}$) is normally distributed with a mean equal to the summation of the variances of each component. Since there are $n = dim(\underline{x})$ components, the mean of $\underline{p}$ is equal to n.

Figure V.3 shows the result of the NEES test for one run of the filter. To pass the NEES test, the expected value of the normalized error should be equal to the dimension of the state space. In the case of this system, the dimension is 15, which is consistent with the result in Figure V.3.

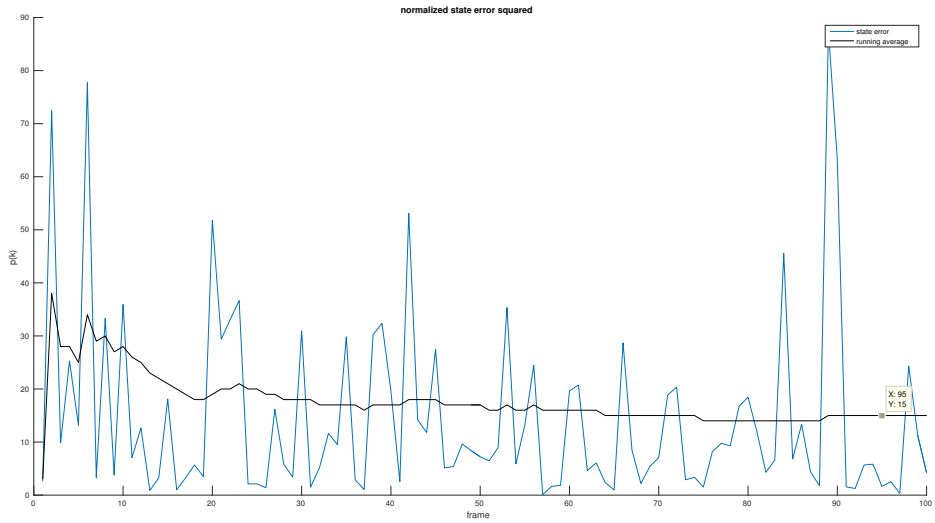

Figure V.3. Normalized error squared (NEES test).

CHAPTER VI

CONCLUSION AND FUTURE WORK

Most of the data and results presented in this thesis are from simulated, rather than experimental, data. The next step in this research is to test the algorithm at the Land, Air, and Space Robotics (LASR) Laboratory. Attitude and position measurements of the SL-8 rocket booster model will be measured with the VICON motion capture system, and will be recorded as "truth". Then, the ICP algorithm will be run given live camera images of the rocket booster. The pose estimates acquired from the algorithm will be compared to the true pose. The Kalman filter presented in Chapter V will be implemented on a system involving both a stationary and moving capture vehicle. To check for filter consistency, Monte Carlo NEES tests for multiple independent runs of the filter should be performed, as in [13]. The equipment that will be used to perform these experiments is shown in Figure VI.1. In this set-up, the SL-8 booster model is attached to a long pendulum that is allowed to translate on low-friction gliders, giving the model a total of 5 degrees-of-freedom. The top of the pendulum is actively controlled, allowing the rocket to exhibit behavior more consistent with a space environment. This behavior is achieved though the use of load cells. When the capture vehicle exerts a force on the rocket body, the rocket body can glide away as if it were in a zero-G environment. This system was developed by Austin Probe [14] at LASR Lab.

Furthermore, the robustness of this algorithm will be put to the test by using a different sensor (with a different resolution and different intrinsic parameters). Given the modularity of the software, a different rocket booster model could also potentially be used.

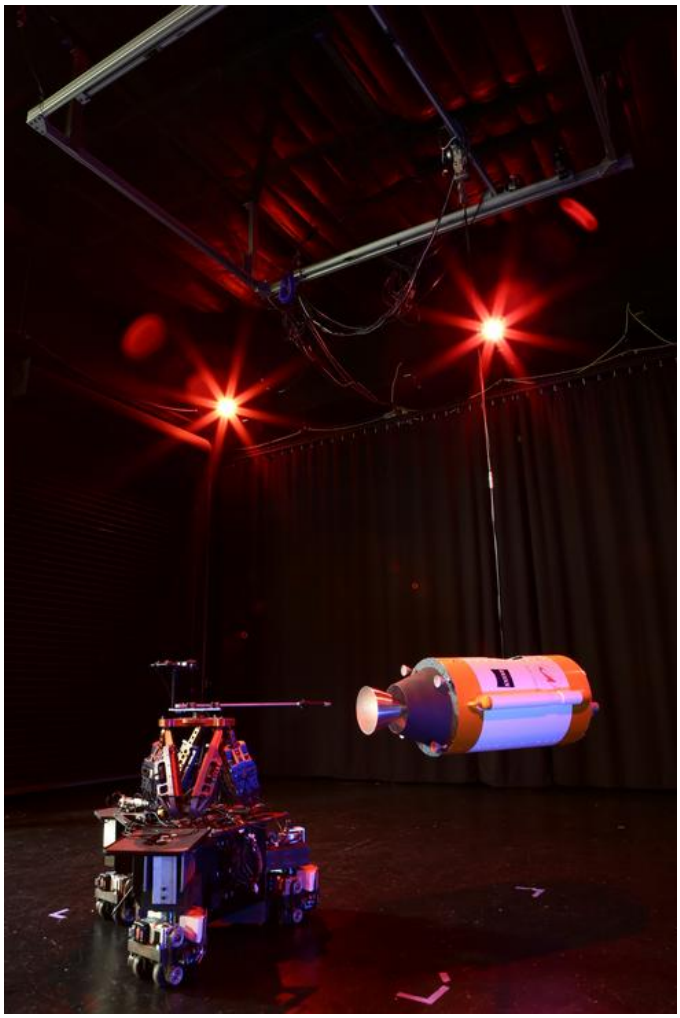

Figure VI.1. Debris capture experiments

Another area that requires further work is the initialization module. The algorithm presented in Section III.D takes between 2 and 3 seconds to determine a pose. This computation time is potentially problematic, especially if the object of interest is rotating at a fast rate. By the time the initialization module computes an estimate, the object might have moved too far for the ICP algorithm to converge to a globally optimal pose. This computation time may be shortened by parallelizing the initialization algorithm in addition to ICP. Another option is to invent another

initialization method entirely, such as one that relies on a kind of binary search that achieves a better solution with each recursive step. All of these future developments, if successful, will be very useful to the overall debris capture system and will make the system much more reliable.

REFERENCES

[1] "Interagency Report on Orbital Debris," Tech. rep., Office of Science and Technology Policy, Houston, November 1995.

[2] "NASA Orbital Debris Programs Office," http://orbitaldebris.jsc.nasa.gov/photogallery/beehives.html, 2012.

[3] Besl, P. J. and McKay, N. D., "A Method for Registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 2, 1992, pp. 239–256.

[4] Ranzuglia, G., "MeshLab," Online. https://sourceforge.net/projects/meshlab/, 2005-2016.

[5] Bouguet, J.-Y., "Camera Calibration Toolbox for Matlab," Online. http://www.vision.caltech.edu/bouguetj/calib_doc/, October 2015.

[6] Tomasi, C. and Manduchi, R., "Bilateral Filtering for Gray and Color Images"," *Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India*, 1998.

[7] Rusinkiewicz, S. and Levoy, M., "Efficient Variants of the ICP Algorithm," *Third International Conference on 3D Digital Imaging and Modeling*, 3DIM, 2001.

[8] Drost, B. et al., "Model Globally, Match Locally: Efficient and robust 3D object recognition," *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference*, 2010.

[9] Ballard, D. H., "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, Vol. 30, No. 2, 1981, pp. 111–112.

[10] Crassidis, J. L. and Junkins, J. L., *Optimal Estimation of Dynamic Systems*, CRC Press, London, 2012.

[11] Hurtado, J. E., *Kinematic and Kinetic Principles*, Publisher: Author, 2012.

[12] Bar-Shalom et al., *Estimation with Applications to Tracking and Navigation*, John Wiley and Sons, New York, 2001.

[13] Bailey et al., "Consistency of the EKF-SLAM Algorithm," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 3562–3568.

[14] Probe, A., *Development of a Robotic Simulation Platform for Spacecraft Proximity Operations and Contact Dynamics Experiments*, Master's thesis, Texas A&M University, College Station, 2013.